D9.4: Final Report on Efficient Integrity Checking

Author(s)	Peter Krauß (KIT), Marion Cadolle Bel (MPCDF), John A. Kennedy (MPCDF), Michal Jankowski (PSNC)
Status	Final
Version	v1.0
Date	04/11/2016

Abstract: With growing scientific communities with large amounts of different data to be stored, mixed storage systems based on disk and tape devices are widely used. These storage systems commonly offer proprietary interfaces with non-uniform feature sets. To ensure data consistency most storage systems rely on the calculation of checksums attached to the stored data objects. In this document we propose a service architecture that is capable of providing a uniform interface to different underlying storage systems to simplify the access to these checksums as well as to other i.e. domain specific metadata records.

EUDAT receives funding from the European Union's Horizon 2020 programme - DG CONNECT e-Infrastructures. Contract No. 654065

Document identifier: EUDAT2020-DEL-WP9-D9.4	
Deliverable lead	KIT
Related work package	WP9
Author(s)	Peter Krauß (KIT), Marion Cadolle Bel (MPCDF), John A. Kennedy (MPCDF), Michal Jankowski (PSNC)
Contributor(s)	
Due date	31/08/2016
Actual submission date	04/11/2016
Reviewed by	Shaun de Witt (UKAEA), Rob Baxter (EPCC), Jos van Wezel (KIT)
Approved by	PMO
Dissemination level	PUBLIC
Website	www.eudat.eu
Call	H2020-EINFRA-2014-2
Project Number	654065
Start date of Project	01/03/2015
Duration	36 months
License	Creative Commons CC-BY 4.0
Keywords	Integrity checking, checksumming, service, HPSS, TSM, storage, archive, long-term preservation

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

In this document we present a new service architecture that enables a user to get easy access to different storage systems such as IBM TSM (see glossary for abbreviations) or HPSS. Hereby we focus on features for integrity checking based on checksumming.

The architecture we propose provides a front-end API following the REST paradigm based on HTTP. This API can then be used to access synchronously retrieve asynchronously calculated and regularly updated checksums of data objects previously stored using widely used storage protocols. Hereby the service works with different backend storage technologies by providing a plugin API invisible to the user.

The architecture was prototypically implemented to work within the B2SAFE context and was successfully tested in defined testing environments.

1. INTRODUCTION

Scientific and cultural organisations, international collaborations and projects have a need to preserve and maintain access to large volumes of digital data for several decades. Existing systems which support these requirements cover the range from simple databases at libraries, to complex multi-tier software environments developed by large-scale scientific communities. All communities can see an increasing volume of digital data that must be stored efficiently and economically. Today, this can involve a dynamically managed combination of storage on magnetic disks and on magnetic tapes.

The EUDAT project[1] is developing an infrastructure for secure and reliable archival storage that functions as a uniform platform for multiple scientific domains and international projects. In this context, access to the actual storage in a datacenter is enabled through an abstracted bit-preservation layer that offers features selected for long-term storage such as special metadata tags. To support this functionality, complex storage systems such as the High Performance Storage System (HPSS) by the HPSS Collaboration[2] or IBM Spectrum Protect for Space Management[3] (SPSM, formerly known as Tivoli Storage Manager/TSM) are often used: they are usually accessible using a proprietary and non-standardized programming interface (API). These APIs require a user to adapt his software in case of a change of the storage provider. In addition, with collaborations growing, and more and more data centres becoming involved, the need to support multiple storage systems arises. Furthermore, in some cases, storage systems which do not provide any integrity information are used, such as storages in the manner of key-value stores.

To cope with these aspects, we propose an architecture that enables the aforementioned functionalities for a variety of different back-ends by abstracting and unifying the underlying storage specific API while maintaining backwards compatibility with regards to the actual data transfer processes. A service consumer (an actual person, another application or another higher-level service) is then able to instantly access integrity information of stored data even if the storage back-end does not provide, or not synchronously, those information natively. Furthermore, a storage back-end can easily changed, i.e., in case of technology upgrades or provider change, without the service consumer's code being reliant on a back-end specific database or API.

1.1. Scope of the Task

In accordance with the project proposal, the work for our subtask 9.3.1 focused on efficient on-site data integrity checks for data based on both tapes and disks, as well as on complex composite storage systems (such as iRODS, HPSS or TSM). The goal of our work was to create a service architecture whose implementation is capable of offering methods for efficient and non-interruptive integrity checks, to higher-level services relying on asynchronous storage systems. Since EUDAT's PID system already handles on-line integrity checking upon data ingest, we specifically investigated options for data stored on tapes, as well as in a continuous manner with fixity checks on a regular basis according to a customer's needs.

Moreover, the service architecture was developed bearing in mind the second half of the project, in which the service will be extended to offer functionality for bringing data online from tape to disk. This yields new requirements with regards to extensibility, scalability and maintainability.

The development process was divided into the following stages:

- Related work and technology exploration
- Architecture development
- Prototypical implementation
- Evaluation

[1] The EUDAT project website, https://www.eudat.eu/eudat-communities-pilots
[2] Website of the HPSS Collaboration, http://www.hpss-collaboration.org/
[3] IBM Spectrum Protect, http://www-03.ibm.com/software/products/de/spectrum-protect-for-space-management

The architecture we proposed is based on a plug-in concept: in addition to a central middleware, the architecture allows to add back-end specific components to abstract the underlying storage layer. This allows us to split the implementation process further into three different phases:

- Base functionality/service core
- Back-end module for SPSM
- Back-end module for HPSS

While the base service middleware was developed at the computing centre at the Karlsruhe Institute for Technology (KIT, Germany), the SPSM back-end was based on previous works (see section 2) and extended at Poznan Supercomputing and Networking Center (PSNC, Poland). The HPSS back-end's implementation has been done by Max Planck Computing and Data Facility (MPCDF, Germany).

In the following sections, we provide an overview of already existing work that relates to our approach. Then, we present the overall service architecture and focus on the implemented prototype, with the two aforementioned back-ends.

2. RELATED WORK

2.1. EUDAT task 7.1.2

One of the results of EUDAT task 7.1.2 “Data Curation and Long-Term Preservation in a Federated Environment” was the pilot implementation of a mechanism performing data integrity checks, like verification of checksums and verification of number of data copies. The implementation targeted at extending the B2SAFE service and later became part of it. This work was described in the EUDAT deliverable D7.2.2 [EUDAT D7.2.2].

From the point of view of our tasks, two policies are interesting:

1. checksum generation policy
2. replica integrity verification policy

Policy (1) enforces the computation of checksums of data stored in B2SAFE as well as its date of calculation. This information is then stored in iCAT in a related handle record.

Policy (2) enforces periodic recomputation of checksums and the verification against previously stored values.

For this present work, we build upon the pilot implementation of this checksum mechanism and shifted the scope from being a B2SAFE-integrated component to a stand-alone service, which does not rely on a specific back-end technology, database, feature set, or API, and which is accessible to any higher-level service capable of performing REST-based communication. In our pilot implementation, we target TSM and HPSS as primarily supported back-ends.

2.2. Data Policy Manager

The Data Policy Manager (DPM) is a software intended to be used by the communities to define policies related to the data, and then to enforce those policies in the EUDAT infrastructure. The core element of it is a central DPM service with a web portal for creating and managing the policies, and a database with policies. The policies are downloaded via DPM agents and enforced by data services. The idea of DPM emerged during the first EUDAT project [DPM 1]. This started within EUDAT 1 and it is still under development within EUDAT2020. The current production version of DPM supports only policies related to data replication for the B2SAFE service. The integration with other services and supporting integrity checking (done within EUDAT task 7.1.2) is planned (see [DPM 2], slide 16).

While DPM focuses on defining and enforcing the policies, the work in task 9.3 focuses on implementation of the integrity checking mechanisms, on the interface to read the checksums and on how to report data corruptions. These two focuses are complementary. DPM and the mechanisms portrayed in the present document may be integrated in future.

3. SERVICE ARCHITECTURE

The system we proposed adds a service interface to a storage system alongside the common Input/Output (IO) channel. While all of the storage back-end IO operations are performed directly via the "data channel", communication with our service is done on a separated channel, referred to as "service channel" (cf. Figure 1).

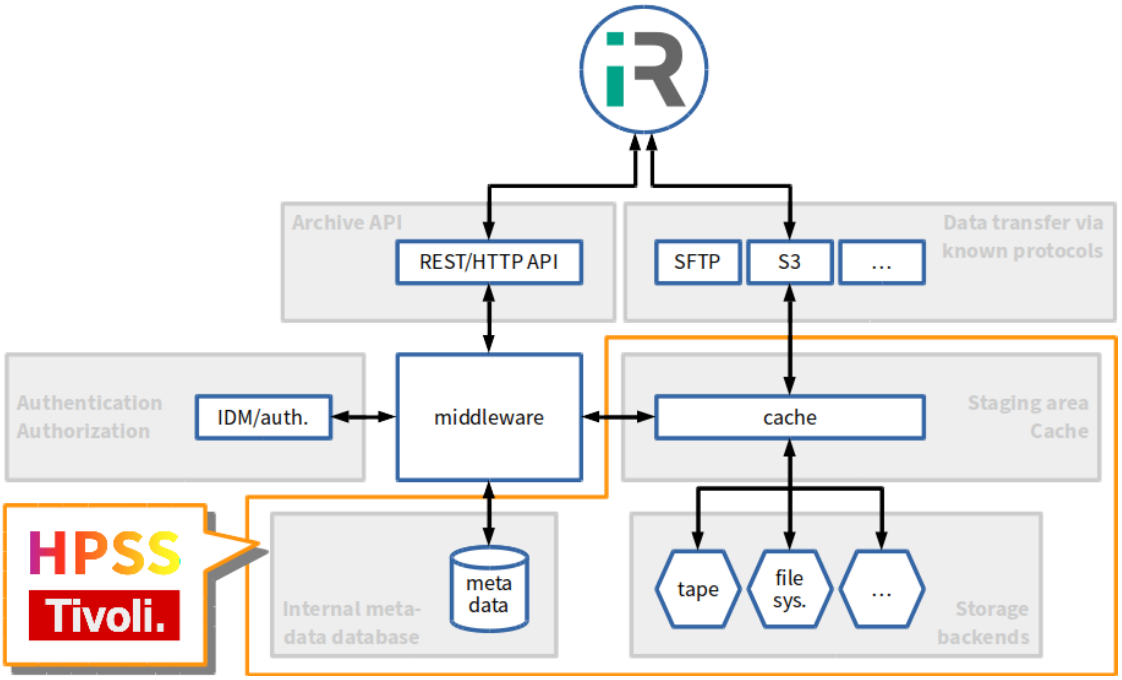

Figure 1: General service architecture

This concept enables existing storage systems and services like B2SAFE to store and access data in the usual way, without interference with our service. However, if the user application chooses to register data using the service channel in the database, they can make use of additional functionalities. The service provides a generic interface, and it is independent of the underlying archive system, due to its functionality to use "back-end specific" modules. Currently, the service enables applications to attach any user defined attributes in the form of metadata, and to access periodically calculated checksums. For a detailed explanation of the latter, as well as an in-depth view of the functionality of the back-end modules for HPSS and TSM, we refer to section 4.

3.1. Front-end API

For communication, an HTTP-based API following the REST paradigm was used. All function calls require a user to send an authentication token within the HTTP header. The API is structured in three namespaces, logically grouping the calls:

1. **/meta:** this namespace contains calls to register data and to attach, read or update metadata. The service keeps several versions of the metadata, allowing an application to retrieve a complete history of a specific metadata set
2. **/data:** this namespace allows to query the service repository. For example, an application can search for data stored within a specific time range, or for data belonging to a particular user
3. **/control:** the namespace provides calls to control the service itself (i.e., calls to stop or restart the service) and to publish service-related status information, like its API version or its current status.

A common workflow usually consists of an authentication call to retrieve a valid authorization token for all further operations, followed by, for example, a call to register a new dataset. This would create a new empty

metadata record attached to the data object, and would return a unique identifier. This identifier is then used for reference in all further calls.

3.2. Metadata Database

It is important for the data centres and storage operators to ensure that the data archive can work autonomously, i.e., without the management software's ability to provide information about the stored data. This includes the ability to provide information, to locate the data in the storage system, to have a persistent identifier (PID) to allow globally unique referencing, and a checksum including its type. Since the location identification or other items of technical metadata may change, it is useful to keep track of the different changes in versioning.

For that purpose, the service includes its own database with the option of freely changing and extending the model for (non-technical) metadata. This can, for example, be useful when the service runs in a stand-alone mode, without connection to a B2SAFE instance. The model can then hold information about administrative regulations, data protection policies and access controls rules, which are specific to each site or each data centre.

Another fit for metadata is the registration of the agreed storage quality. Data centres like the Steinbuch Centre for Computing (SCC) at the KIT charge for the use of resources. Some data may be stored in different protection classes, i.e., using a different number of tapes located on- or(and) off- site. Also, one must always know who the owner of the stored data is. The person or organization responsible for that information, or a contact to whom questions can be sent, is registered using the service channel of the archive interface.

Finally, the service's metadata records can be extended to record the physical location of a stored data set. This will be of great value when it comes to the second half of our project, which focuses on bringing data on-line stored on slow storages (like tape systems).

3.3. Authentication and Authorization

The service authorization mechanism used is also based on HTTP. Currently, each authenticated user is granted an API token that has to be sent with each API call. For that purpose, we are following a standard defined in [RFC7519], where the authentication information is securely encoded in the API call's header, using common web technology.

For now, no fine-grained permission system is implemented, due to the whole infrastructure currently being controlled by a single service provider making a strict permission system unnecessary. The architecture however allows this to be added in the future.

3.4. Plug-in Interface

One of the main purposes of the system is to implement different back-end specific modules. These modules are connected using a low-level plug-in interface in addition to the front-end API. This interface offers additional system-specific functions needed for proper operation of the system, but not accessible by the user, even though it is required by the back-end modules. The API is for example used to (periodically) request a list of data objects to be checked for integrity based on user defined criteria (referred to as "candidates") as well as their physical position on a storage device. A storage-specific module then follows the workflow described in the following sections (section 4). In the current implementation, with the service being connected to a B2SAFE instance, the storage back-ends are configured to obtain the candidates list from the B2SAFE's iRODS database. After checksum calculations, the back-end sends back the checksums for validation to the middleware. In case of non-matching checksums, the middleware can be configured to trigger different actions to notify the service operator that there is a problem or an error on a specific (list of) file(s). In the current prototype, the default action is to notify an administrator by email. In a production environment, more complex policies need to be defined and followed, as outlined in the section 5.

4. MIDDLEWARE IMPLEMENTATION DETAILS

The middleware component cooperates with metadata database that contains information on all stored files, for example: checksum values, algorithm and date of its last computation, or information such as “Is the value a pattern set by the user/Was the value computed in the system/Was the last check successful?...”

For the current pilot implementation, we use the iCAT database, which is part of the B2SAFE service.

The periodic checksumming workflow is as follow:

- The **candidates collector** selects a list of candidate files (i.e, the ones which should have the checksum computed or recomputed in the nearest future) from the metadata database. The selection is based on the time of the last computation, or if the checksum value is lacking. The output of the module is a list of candidate files. A record of that contains the path, the type of checksum algorithm (MD5, SHA256, etc.) and a priority (explained below)
- The **cache recaller and checksum calculator** processes the list of those candidate files and decides which checksums shall be calculated, prepares and performs the calculation. It outputs the outcome to a results file. A record of the results file contain the file path, value of the checksum, checksum algorithm type and time of the computation. The recaller and calculator are a pluggable, storage-specific module. It has many implementations, depending on the characteristics and interface of the storage, e.g.,
  - in the case of a disk storage, the checksums may be computed directly on the storage
  - in the case of a hierarchical storage, like Spectrum Protect (TSM) or HPSS, the files must be staged in an effective way (using native mechanisms).
- The **checksum verifier** reads the results file and checks the computed checksums against the values which were previously stored in the metadata database.

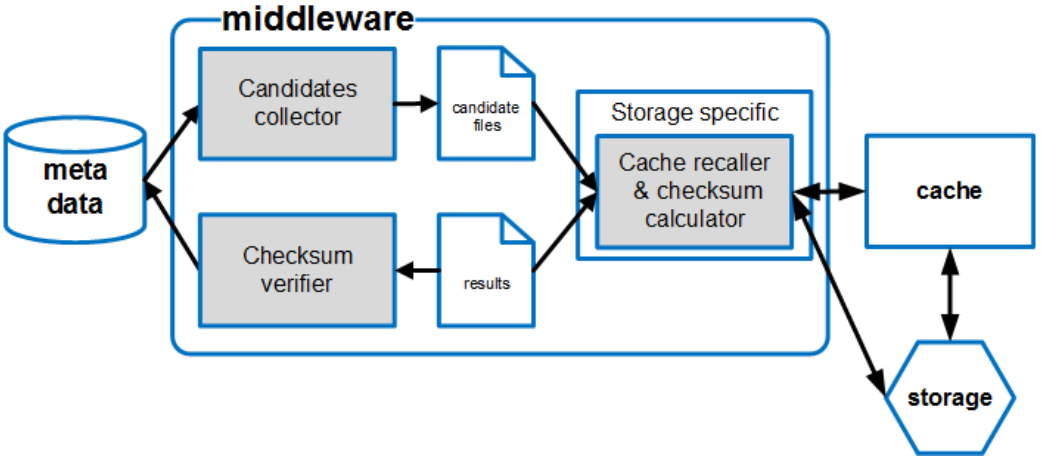

Figure 2: Middleware details

The length of the checksum verification cycle (the time between following computations and verifications of a single file’s checksum) is measured in days. In the current implementation it is a parameter defined by the site administrator. The priority of checksum recalculation is the number of days since the last calculation, minus the length of the cycle. Thus, the positive priority means the checksum must be recalculated as soon as possible, and the negative priority means the checksum may be recalculated in the future. The candidates collector selects all the files with a positive priority, and the files with a negative priority close to 0. The cache recaller and checksum calculator module decides which files with negative priorities will be processed optimizing the process, especially the cache recall. For example, negatives located on the same tape as some positives may be processed (to avoid mounting the same tape day after day) while other negatives may be ignored (until the process goes to the next tape, for example).

4.1. General Integration

The storage specific back-ends are based on bash shell scripts and back-end specific command line tools. However, other executables can be used, as the interfaces to the other modules (candidates collector and checksum verifier) are text files. This includes different scripting languages or precompiled binaries.

4.2. Spectrum Protect (TSM) for Space Management

IBM Spectrum Protect (formerly TSM) for SPSM plays the role of HSM. It manages moving the less frequently used and bigger files from cache (disk storage) to a cheaper media (tapes). A file goes into the cache before being stored. Then, it may be copied to a tape. Finally, it may be swept from the cache, and only a small stub file is left in the cache. The decision is made up by the Space Management and is based on the file size, its access time and on the free space left in the cache.

For reading, the file must be located in the cache. If the file was swept from the cache, an ad hoc reading request causes queuing cache recall request. Typically the queue is of FIFO type, so this mechanism is not effective for reading multiple files, since it causes multiple tape loads and rewinds. In the case of multiple files to be recalled, a procedure called “tape optimized recall” should be performed. In the first step of this procedure, requests are grouped by tapes, then sorted by order of appearance on each tape, and, finally, a list of such ordered files with identifiers of tapes and offsets is produced. In the next step, the recall is performed using this information.

The Space Management back-end, compatible with described before periodical checksumming workflow, was initially developed as an addition to the B2SAFE service. It is included in the B2SAFE repository master branch and officially released. It is called “TSM back-end” due to historical reasons. It implements an algorithm based on a modified tape optimized procedure. The algorithm contains and implements these steps with respect to the files from the candidates list:

- checks which files are available in the cache and compute checksums for them first
- prepares the recall of the rest of the files
- analyses the result of the above, identify tapes that contain only files with negative priorities, remove files located on such tapes from the list to recall
- prepares the recall of the rest of the files
- performs the optimized recall computes the checksums and write them to the output file.

A simple test was performed in order to prove the above approach. We randomly selected 50 000 files from a production system. In the first step we checked the status of these files and we got:

- 12 984 files migrated from the disk cache
- 37 016 files available in the cache (TSM status resident or pre-migrated); out of this number:
  - 35 491 small files (of size under 1MB); these files would never be migrated (according to the migration policy of the tested system),
  - 1 525 big files (i.e., 3% of the initial number); these files might be migrated by the Space Management at any time.

The above numbers show that computing the checksums first for files initially available in the cache is reasonable from the point of view of efficiency - otherwise recalling these 3% might be necessary.

The aim of the next step was to measure and compare effectiveness of ordered and unordered recall. We decided to use files of size 10-11 MB, as it is a relatively small size for a file intended to be stored on a tape system. Hence, we expected to see a considerable difference between ordered and unordered recall.

We selected files of that size out of the migrated files from our test set and we obtained 151 files with a total size of 154 GB. As it occurred, the files were distributed between 51 tapes. First, the files were recalled to the cache sequentially, in the order of appearance. Then the files were removed from the cache, and a “tape-optimized recall” was performed. The measured times of both recalls are indicated in the table below.

Test	Test type:	Time to complete
1	unordered recall	312 min. 13 sec.
2	ordered recall	38 min. 20 sec.

As expected, the ordered recall is much more effective. One reason for that is that the tested environment had five drives and the ordered recall worked with up to five parallel threads (depending on the actual usage of the production system), while the unordered recalls were done sequentially in a single thread. Note, however, that the unordered recall lasted much longer than five times of the ordered recall: this would last longer, even on a system with single drive.

The general conclusion is that the presented above algorithm is optimal for the specificity of SPSM storage.

4.3. HPSS

HPSS is the result of over a decade of collaboration among five Department of Energy laboratories and IBM, with significant contributions by universities and other laboratories worldwide. HPSS is a software that manages petabytes of data on disk and robotic tape libraries, providing highly flexible and scalable hierarchical storage management that keeps recently used data on disk, and less recently used data on tape. This way, HPSS easily meets excellent demands of total storage capacity, file sizes, data rates, and number of objects stored.

The HPSS back-end was developed after an extensive evaluation of the HPSS client tools HSI, HTAR and FUSE.

- **HSI:** Hierarchical Storage Interface - A powerful and flexible command line tool for accessing HPSS resources. HSI provides an interface similar to many FTP clients, allowing traversal of the filesystem and data management such as puts and gets; while also providing HPSS specific functionality. HSI can be used in either interactive or batch modes
- **HTAR:** ("HPSS Tape Archiver") is a TAR-like utility program that makes TAR-compatible archive files but with HPSS support and enhanced archive-management features
- **FUSE:** A virtual filesystem which enables local users to access remote HPSS based storage in a POSIX like way.

Each of these tools was evaluated considering the specific use-case of checksum generation, and also the wider use-cases which are relevant to WP9.3. Indeed, tools are needed to manage mass data ingest/extraction to/from tape and also to provide a good layer of abstraction between user requests and the underlying archive system.

Following the evaluation, HSI was chosen as the base tool for the generation and re-evaluation of checksums in HPSS. The script is designed to wrap the HSI command line client and to generate a checksum on a “file-by-file” basis. This ensures that the script has a simple workflow and that it can generate a different checksum type for each candidate file. However, it also leads to a performance penalty, as each checksum verification is coupled with an authentication overhead.

The HPSS checksum verification script was written so that it is compatible with the B2SAFE checksum verification workflow. Moreover, the script has been accepted into B2SAFE, and is now available for those HPSS sites which wish to enable integrity checking.

Once a working script was available, a performance evaluation was undertaken to find ways to speed up the checksum verification process. Several tests were undertaken to understand the impact of the file ordering, authentication overheads, and the use of bulk commands, where many files are processed at once.

The results from these tests can be best explained by considering the following three test cases:

- Sequential checksum generation using a random file list; no respect for tape or position on tape (one HSI call per file);

- Sequential checksum generation using an ordered file list; files grouped by their tape and ordered by their position on the tape (one HSI call per file);
- Bulk command; making use of HSI's ability to process a whole list of files, whereby it orders these internally by tape and their position on tape (one single HSI call for all files).

The tests were performed using one hundred 10MB files, which were staged to HPSS and cleared from the disk cache before each test (thus ensuring a level playing field for the tests). The following table outlines the test results, showing which methods address the issues of file ordering and authentication overhead:

Table 1: Test cases and results

Test	Test type	Time to complete	Performance impacts (positive/negative)
1	Sequential - Random file order	~ 70 minutes	No File Ordering Authentication Overhead
2	Sequential - Ordered files	~ 21 minutes	File Ordering Authentication Overhead
3	Bulk request - process all files at once	~ 9 minutes	File Ordering No Authentication Overhead

The tests were undertaken on a test system with a single tape drive. Therefore, the tests will tend to heavily penalise tape mounts. An investigation of the files' list showed that 24 tape mounts (swaps) were needed for the random ordered file test (test 1) while only 4 tape mounts were required when the files were ordered (tests 2 and 3). The "Performance impacts" column highlights the positive/negative impacts with respect to file ordering and authentication when using the different methods. In further details, one may notice that:

- Test 1 is by far the slowest: due to the random file ordering, many tape mounts are needed and the sequential checksum generation leads to an authentication overhead for each file;
- Test 2 shows a large improvement in comparison to test 1, which is due to the file ordering: this leads to far fewer tape mounts. The authentication overhead is, however, still present;
- Test 3 performs the best, since the tape mounts are reduced due to the HSI client ordering the file list internally, and the authentication overhead is reduced, since only a single client call is required.

These tests show that using bulk methods and file ordering is highly recommended when managing large tape-based data sets. File ordering leads to improved read speeds and a reduced amount of tape mounts. Bulk methods often provide in-build file ordering and also reduce any authentication overhead (a single method call is better than several hundreds).

5. POLICY FOR INTEGRITY CHECKING AND REPORTING CORRUPTED DATA

In addition to providing a technical solution for data integrity checking and the reporting[4] of corrupted data, we also need to define policies in these two areas. These policies should be defined in a EUDAT wide context, linking to the work of the Data Policy Manager as described in the initial EUDAT project [Deliverable 7.2.2].

The policies should allow us to:

1. identify data sets which need periodic integrity checks (and define the period)
2. clearly define the actions to take upon the discovery of corrupted data.

The concept of Storage Classes exists within EUDAT [WP6DEL] and can be further developed to contain classes which include periodic integrity checking. This would allow projects and communities to categorize their data, ensuring that specific/precious data sets have multiple tape resident copies, the latter being periodically checked for data integrity.

A process for managing instances of data corruption requires both input from the infrastructure and service provider(s) and the supported data communities. A set of policies should be developed that clearly defines the roles of each party and also highlights the different options with respect to integrity checking, the implications, and the associated costs.

These policies need to define:

- contact partners - on both sides, Data Community and Infrastructure provider
- processes to follow upon the identification of corrupted data (in no specific order):
  - deletion of data records (including PID updates)
  - auto-correction of data, retrieve redundant copy and apply updates
  - reporting and recording process (email to community contacts but also PID catalogue updates etc.)
- the frequency of the integrity checking

Tentative discussion about these policy issues have begun within the EUDAT project, and are expected to continue into the second half of the present project.

[4] Methods for reporting data corruption have been considered, however, further testing/evaluation work is needed.

6. INFRASTRUCTURE CONSIDERATIONS - THE "COST" OF FIXITY CHECKING

Implementing a system to check data integrity on a regular basis will obviously lead to an increase in load on the archive systems. To gain some insights into the effective "cost" of this, we made a ballpark estimation considering the expected read speed of the archives, and the required daily data processing rate an example EUDAT integrity policy would lead to.

An estimate of the effective read rate when reading data from a single tape (or drive) of type LTO6 2.5 TB (or LTO5 1.5 TB) is of the order 3 to 4 TB per day. This is a little below the theoretical maximum considering our real experience when reading a lot of "small" files. Considering the type of fixity checking policy which has been proposed for EUDAT, i.e., yearly checksum verification, if we store 1 PB of data in the EUDAT CDI, we would then need to read and checksum 2.74 TB per day. Thus, per PB of data (with yearly fixity checking), we would need one dedicated tape drive. Note that we have not taken into account anything regarding on how long it would take to calculate the checksums, to swap tapes, etc. (these issues may also lead to overheads). If we assume that requirements for integrity checking do not exceed 12 months periodicity, the data center costs are estimated to comprise investment and operational costs of 1 to 2 additional tape drives (including data center integration). At KIT these costs amount to 5.000 € to 7.000 € which results in additional costs of 5 to 7 euros per year per TB stored.

This ballpark estimation shows that periodic integrity checking is a costly exercise. Enabling this at a scale of many Petabytes will require not only additional services, such as the one we are developing, but also increases in the existing archive infrastructure. More tape drives will need to be purchased if we want to ensure that the integrity checking will not impact the archive users. This may make periodic integrity checks for the whole data set prohibitive. A viable alternative may be to define a storage quality which contains periodic checksumming. Not all the data sets will need to have this storage quality, which means that the infrastructure costs could be kept down. However, data sets which are identified as being extremely precious may be assigned a higher storage quality to ensure that they are regularly checked.

7. CONCLUSION

Thanks to this prototypical implementation, we can demonstrate that the architecture we presented enables easy-to-use and uniform fixity checking in archive resources. Those are currently TSM/HSM and HPSS. The code is now part of the last B2SAFE release. In the near future, we will continue the work on the prototype to further decouple it from the underlying storage technology as well as the iRODS middleware. During the past months, it has become increasingly clear that we need continued dialog with archive vendors in order to ensure that the integrity checking becomes part of the archive system itself. This will avoid the administrative overhead of a standalone service structure. However, a uniform standardized API will then be required for better user acceptance, and to avoid the phenomenon known as “vendor lock-in”. This phenomenon states, that a user is de-facto forced to use a certain technology provider due to incompatibilities of interfaces and data formats making it impossible or not economically reasonable to change the provider.

Because of the different associated costs, we need to consider different storage quality classes. Some projects will be willing to pay more for integrity checking, some may simply want a second/third/fourth copy of the data. This aspect of trading performance against money against data security has to be considered by the service provider as well as the end user.

Beside the technical aspects of integrity checking, policy discussions have started: these are needed to ensure proper storage qualities suitable for different user groups needs. This needs further definitions, clarifications, and that procedures are developed to know what to follow and to do when some corruption in the data is observed.

ANNEX A. HTTP-HPSS PROXY

An HTTP-HPSS proxy (developed by Christof Hanke, MPCDF) was released as an open source package. This proxy was initially developed for the internal use at MPCDF. However, it became of direct relevance to our work in EUDAT WP9 and, as such, was released as an open source software for general use and further potential developments. MPCDF did an initial evaluation of the HTTP-HPSS proxy as a possible generic interface to HPSS, and the results were quite encouraging.

The HTTP-HPSS component offers easy HTTP-based access to the HPSS API. It represents a significant simplification for the user. The component is written in C with bindings for Python. The component's core is an HTTP listener based on "libevent" running on a machine directly connected to HPSS. The listener service uses HPSS's C API for any operations with HPSS (on the back-end). Whereas the HPSS API only offers low level functions (such as to read a slice of a file, get an access handle or a pointer to a file system iterator), the front-end of the service offers higher level operations (for example, to up-/download complete files). The proxy authenticates itself against HPSS on initialization, and it keeps a session open throughout its lifetime, leading to improved performance. Additionally, the HTTP-HPSS proxy may be configured to expose only a subset of possible functions to the user. This means that a proxy could be deployed to only allow read and list access to the underlying HPSS system. In conclusion, this proxy simplifies access to an underlying HPSS archive, speeds up actions with the archive, and allows for configurable/secure deployments.

ANNEX B. GLOSSARY

Term	Explanation
IBM	International Business Machines (Corporation)
HPSS	High-Performance Storage System
HSM	Hierarchical Storage Manager
SPSM	Spectrum Protect for Space Management
TSM	Tivoli Storage Manager
DPM	Data Policy Manager
Checksum	small-size datum from a block of digital data for the purpose of detecting errors which may have been introduced during its transmission or storage.
API	Application program interface consisting of a set of routines, protocols, and tools for building software applications.
Rest (API)	Architectural style: RESTful implementations make use of standards, such as HTTP, URI, JSON, and XML. Unlike SOAP-based web services, there is no "official" standard for RESTful web APIs.
HTTP	The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems.
iRODS	Integrated Rule-Oriented Data System: distributed data-management system for creating data grids, digital libraries, persistent archives, and real-time data systems.
PID	Persistent identifier associated to a digital object or to a whole collection
iCAT	iRODS Catalogue -metadata database for iRODS
metadata	Additional information added to a data record to describe the actual data. Commonly metadata includes information about the data's origin, the data's lifecycle or the data's content.
authentication token	An authentication token is a digital signature generated from authentication information to identify a user or entity.
FIFO	First In First Out
HSI	Hierarchical Storage Interface
HTAR	HPSS Tape Archiver
FUSE	Filesystem in Userspace
DPM	Data Policy Manager

ANNEX C. REFERENCES

[EUDAT D7.2.2] EUDAT deliverable D7.2.2: "Managing Data Curation and Long-Term Preservation in a Federated Environment (final version)".

[DPM 1] Willem Elbers, Adil Hasan, Data Policy Manager, presentation during 3rd EUDAT Conference, 25.09.2014, available at https://www.eudat.eu/sites/default/files/WillemElbers.pdf

[DPM 2] Mark van de Sanden, EUDAT Services, presentation during EUDAT User Meeting, 22-23 June 2016, Barcelona, available at https://www.eudat.eu/sites/default/files/EUDAT%20Services%20Overview_vandeSanden.pptx

[RFC7519] Internet Engineering Task Force RFC7519, "JSON Web Token (JWT)", M. Jones, Microsoft, J. Bradley, Ping Identity, N. Sakimura, NRI, May 2015, https://tools.ietf.org/html/rfc7519

[WP6DEL] Johannes Reetz (RZG), Giovanni Morelli (CINECA), Cristina Manzano (JUELICH), Kostas Kavoussanakis (UEDIN), Pascal Dugenie (CINES), David Vicente (BSC), Urpo Kaila (CSC) ; Claudio Cacciari (CINECA), Heinrich Widmann (DKRZ), Florian Kaiser (RZG), Robert Verkerk (SURFsara), Dejan Vitlacil (SNIC), Carl Johan Hakansson (SNIC), Pietari Hyvärinen (CSC), May 2015, "EUDAT D6.4 Final Report on Operations; v1.0"

